PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-014826
(43)Date of publication of application : 18.01.2002

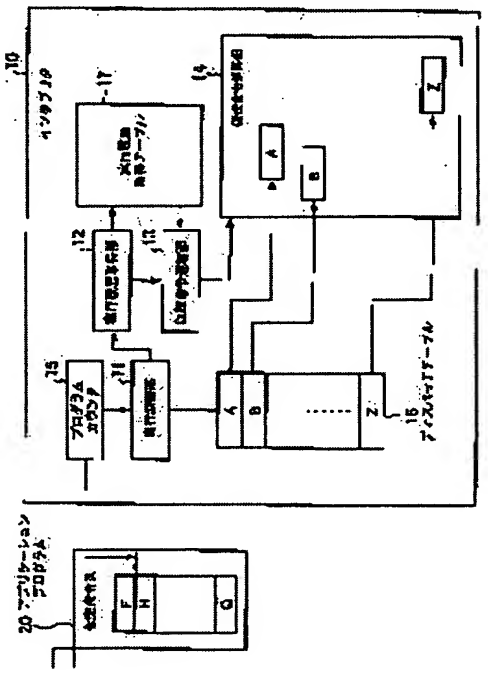
(51)Int.Cl. G06F 9/45

(21)Application number : 2000-196871
(22)Date of filing : 29.06.2000

(71)Applicant : TOSHIBA CORP
(72)Inventor : SAKAI RYUJI

(54) INTERPRETER AND METHOD FOR EXECUTING ITS PROGRAM

(57)Abstract:
PROBLEM TO BE SOLVED: To provide an interpreter capable of realizing acceleration equivalent to that of JIT compiler with a simpler mechanism without scarifying the starting time of a program.
SOLUTION: An execution control part 11 obtains, from a dispatch table 16, the entry address of a virtual instruction processing part 14 for processing a virtual instruction 21 indicated by a program counter 15 allows the address to jump, and notifies an execution history obtaining part 12 of which virtual instruction is the target for execution control. On the other hand, the execution history obtaining part 12 makes a pair of the virtual instruction and the previously communicated virtual instruction, and monitors the execution frequency by counting the number of times of appearance by using an execution history obtaining table 17, and notifies a virtual instruction connecting part 13 of the connection of the pair whose execution frequency is high as one virtual instruction. Then, the virtual instruction connecting part 13 executes the reconstitution of a virtual instruction series in order to connect the communicated pair as one virtual instruction.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2002－14826
（P2002－14826A）
(43)公開日　平成14年１月18日(2002.1.18)

(51)Int.Cl.'	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　9/44	３２０Ｃ　５Ｂ０８１

審査請求　未請求　請求項の数９　ＯＬ　（全 10 頁）

(21)出願番号　特願2000－196871(P2000－196871)

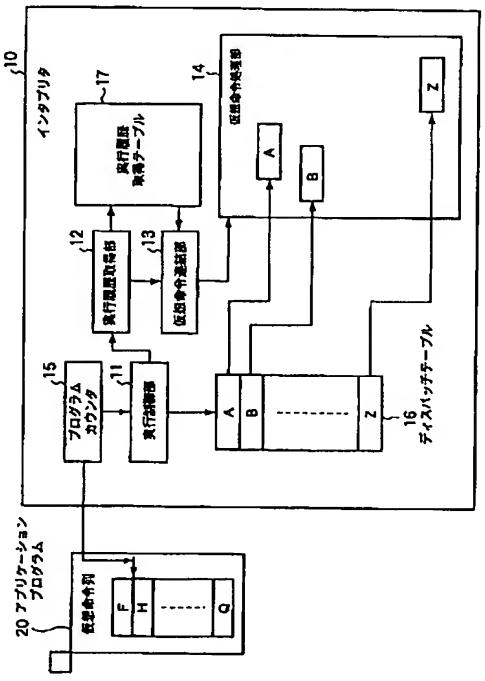
(22)出願日　平成12年６月29日(2000.6.29)

(71)出願人　000003078
株式会社東芝
東京都港区芝浦一丁目１番１号
(72)発明者　境　隆二
東京都青梅市末広町２丁目９番地　株式会社東芝青梅工場内
(74)代理人　100058479
弁理士　鈴江　武彦　（外６名）

Ｆターム(参考)　5B081 DD01

(54)【発明の名称】　インタプリタおよびそのプログラム実行方法

(57)【要約】
【課題】プログラムの起動時間を犠牲にすることなく、ＪＩＴコンパイラより単純な仕組みで同等の高速化を実現するインタプリタを提供する。
【解決手段】実行制御部１１は、プログラムカウンタ１５が指し示す仮想命令２１を処理するための仮想命令処理部１４のエントリアドレスをディスパッチテーブル１６から取得してジャンプさせるとともに、どの仮想命令について実行制御したのかを実行履歴取得部１２に通知する。一方、実行履歴取得部１２は、この仮想命令と一つ前に通知された仮想命令とをペアにし、その出現回数を実行履歴取得テーブル１７を用いてカウントすることにより実行頻度を監視して、実行頻度の高いペアを１つの仮想命令として連結すべくその旨を仮想命令連結部１３に通知する。そして、仮想命令連結部１３は、その通知されたペアを一つの仮想命令に連結すべく仮想命令列の再構成を実行する。



【特許請求の範囲】

【請求項１】　複数の仮想命令からなる一連の仮想命令列の実行履歴を取得する実行履歴取得手段と、
前記実行履歴取得手段により取得された実行履歴から実行頻度の高い前記一連の仮想命令列を検出する検出手段と、
前記検出手段により検出された前記一連の仮想命令列を一括処理する処理部を新たに生成して未定義の空き仮想命令を割り当てるとともに、その割り当てた空き仮想命令として解釈実行されるように前記一連の仮想命令列を再構成する仮想命令連結手段と、
を具備することを特徴とするインタプリタ。

【請求項２】　前記仮想命令連結手段は、新たに生成して未定義の空き仮想命令を割り当てた処理部が予め定められた条件を満たすときに、その処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列をさらに再構成する手段を有することを特徴とする請求項１記載のインタプリタ。

【請求項３】　複数の仮想命令からなる一連の仮想命令列の実行履歴を取得する実行履歴取得手段と、
前記実行履歴取得手段により取得された実行履歴から実行頻度の高い前記一連の仮想命令列を実行中に検出する検出手段と、
前記検出手段により検出された前記一連の仮想命令列を一括処理する処理部を新たに生成するとともに、その生成した処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列を再構成する仮想命令連結手段と、
を具備することを特徴とするインタプリタ。

【請求項４】　前記アドレス呼び出し用の仮想命令は、未定義の空き仮想命令を用いて設けられるものであることを特徴とする請求項２または３記載のインタプリタ。

【請求項５】　最適化の対象を限定する手段を有することを特徴とする請求項１、２、３または４記載のインタプリタ。

【請求項６】　仮想命令と処理部とを対応づけるディスパッチテーブルを最適化の対象別に備えて、それぞれに固有の最適化を行うことを特徴とする請求項１、２、３または４記載のインタプリタ。

【請求項７】　複数の仮想命令からなる一連の仮想命令列の実行履歴を取得し、
この取得した実行履歴から実行頻度の高い前記一連の仮想命令列を検出し、
この検出した前記一連の仮想命令列を一括処理する処理部を新たに生成して未定義の空き仮想命令を割り当てるとともに、その割り当てた空き仮想命令として解釈実行されるように前記一連の仮想命令列を再構成することを特徴とするインタプリタのプログラム実行方法。

【請求項８】　新たに生成して未定義の空き仮想命令を割り当てた処理部が予め定められた条件を満たすときに、その処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列をさらに再構成することを特徴とする請求項７記載のインタプリタのプログラム実行方法。

【請求項９】　複数の仮想命令からなる一連の仮想命令列の実行履歴を取得し、
この取得した実行履歴から実行頻度の高い前記一連の仮想命令列を検出し、
この検出した前記一連の仮想命令列を一括処理する処理部を新たに生成するとともに、その生成した処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列を再構成することを特徴とするインタプリタのプログラム実行方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】この発明は、仮想命令を用いて記述されたプログラムを解釈実行するインタプリタおよびそのプログラム実行方法に係り、特に、プログラムの起動時間を犠牲にすることなく、ＪＩＴコンパイラより単純な仕組みで同等の高速化を実現するインタプリタおよびそのプログラム実行方法に関する。

【０００２】

【従来の技術】インタプリタとは、仮想的な命令をソフトウェアで解釈し、実行するプログラムのことである。インタプリタは、命令を解釈し実行する一つのループによって実装される。このループの中には、仮想命令をメモリからロードして命令の種類を判別し、その命令に相当する処理へ分岐する命令ディスパッチ処理の部分と各命令毎に命令の機能を実行する部分とがある。

【０００３】インタプリタによるプログラムの実行は、同等の処理をターゲットのマイクロプロセッサの機械語命令にコンパイルして動作するプログラムと比較すると、数倍から十数倍遅くなる。そこで、インタプリタによる実行を少しでも高速にするために、実行する仮想命令を、予め決めれらた別の命令に変更したり、複数の仮想命令を同等の処理を行う一つの仮想命令に置き換えることが行われている。また、一連の仮想命令列を、プログラム実行時に機械語命令列にコンパイルしてから実行するＪＩＴ（Ｊｕｓｔ　Ｉｎ　Ｔｉｍｅ）コンパイルという手法も存在する。

【０００４】

【発明が解決しようとする課題】ところで、この仮想命令を別の命令に書き換えながら実行する方法は、予め特定の仮想命令や仮想命令のペア、あるいは仮想命令のシーケンスを、特定の仮想命令（空き仮想命令を使うこともある）に置き換えるための規則と仕組みとをインタプリタに与えておく必要があるので、仮想命令の定義できる範囲でしか書き換え規則を定義できないので、高速化の観点からは、それほど大きな効果がなかった。

【０００５】一方、ＪＩＴコンパイルによる高速化は、

仮想命令列より機械語命令列の方がサイズが大きくなるため、多量のメモリを必要とするので、メモリサイズの小さい組み込み機器などでは使えないことが多い。また、各処理単位の最初の実行時にＪＩＴコンパイルする場合、プログラムの起動時間が長くなってしまうためにＪＩＴコンパイラの使用をやめてしまうケースも多い。

【０００６】また、メモリサイズの小さなシステムでは、全ての処理をＪＩＴコンパイルするのではなく、実行頻度の高い処理のみをＪＩＴコンパイルすればよいが、実行頻度を求めてＪＩＴコンパイルする仕組みが、多くのメモリを使用してしまうという問題もある。

【０００７】この発明はこのような事情を考慮してなされたものであり、プログラムの起動時間を犠牲にすることなく、ＪＩＴコンパイラより単純な仕組みで同等の高速化を実現するインタプリタおよびそのプログラム実行方法を提供することを目的とする。

【０００８】

【課題を解決するための手段】前述した目的を達成するために、この発明のインタプリタは、拡張性の確保等のために一般的に用意される空き仮想命令を利用して、プログラムの実行中に実行頻度の高い仮想命令列を動的に１つの仮想命令として再構成していくようにしたものであり、そのために、複数の仮想命令からなる一連の仮想命令列の実行履歴を取得する実行履歴取得手段と、前記実行履歴取得手段により取得された実行履歴から実行頻度の高い前記一連の仮想命令列を検出する検出手段と、前記検出手段により検出された前記一連の仮想命令列を一括処理する処理部を新たに生成して未定義の空き仮想命令を割り当てるとともに、その割り当てた空き仮想命令として解釈実行されるように前記一連の仮想命令列を再構成する仮想命令連結手段とを具備することを特徴とする。

【０００９】この発明のインタプリタにおいては、実行頻度の高い仮想命令列について、その実行に必要となるｎ回のループを１回のループで済むように動的に再構成を行うことにより、プログラムの起動時間を犠牲にすることなく、ＪＩＴコンパイラより単純な仕組みで同等の高速化を実現することを可能とする。

【００１０】また、この発明のインタプリタは、前記仮想命令連結手段が、新たに生成して未定義の空き仮想命令を割り当てた処理部が予め定められた条件を満たすときに、その処理部をアドレス呼び出し用の仮想命令から呼び出すべく解釈実行されるように前記一連の仮想命令列をさらに再構成する手段を有することを特徴とする。

【００１１】この発明のインタプリタにおいては、有限である空き仮想命令を有効に利用することを可能とす。

【００１２】また、この発明のインタプリタは、前記アドレス呼び出し用の仮想命令が、未定義の空き仮想命令を用いて設けられるものであることを特徴とする。

【００１３】この発明のインタプリタにおいては、アドレス呼び出し用の仮想命令が予め定義されていない場合であっても、同手法での最適化を行うことを可能とする。

【００１４】また、この発明のインタプリタは、最適化の対象を限定する手段を有することを特徴とする。

【００１５】この発明のインタプリタにおいては、たとえばプログラムの実行計画などに対応させて最適化の効果を得やすくすることを可能とする。

【００１６】また、この発明のインタプリタは、仮想命令と処理部とを対応づけるディスパッチテーブルを最適化の対象別に備えて、それぞれに固有の最適化を行うことを特徴とする。

【００１７】この発明のインタプリタにおいては、たとえばメソッド、関数または処理単位等ごとに最適化を行うことを可能とする。

【００１８】

【発明の実施の形態】以下、図面を参照してこの発明の一実施形態を説明する。

【００１９】図１は、この発明の実施形態に係るインタプリタのプログラム実行方法を説明するための概念図であり、インタプリタ１０は、実行制御部１１、実行履歴取得部１２、仮想命令連結部１３および仮想命令処理部１４の各処理部と、プログラムカウンタ１５、ディスパッチテーブル１６および実行履歴取得テーブル１７の各データ部とにより、アプリケーションプログラム２０の仮想命令列２１を解釈実行しながらその最適化も動的に行っていく。

【００２０】なお、この図１に示すインタプリタ１０およびアプリケーションプログラム２０は、ハードディスク等の外部メモリからコンピュータの主メモリにロードされたものであり、たとえば後述する仮想命令列２１の書き換え等は、あくまで主メモリ上だけで行われるものである。

【００２１】実行制御部１１は、プログラムカウンタ１５が指し示す仮想命令２１をロードして、その仮想命令２１を処理するための仮想命令処理部１４のエントリアドレスをディスパッチテーブル１６から取得し、その取得した仮想命令処理部１４のエントリアドレスに制御を移す（ジャンプする）。

【００２２】このディスパッチテーブル１６は、各仮想命令と仮想命令処理部１４のエントリアドレスとを対応づけるためのものであるが、図２に示すように、予め定義された既存の仮想命令のエントリアドレスを格納する領域に加え、拡張性を確保する等のために、新設の仮想命令のエントリアドレスを格納するための空き領域を備えている。また、このインタプリタ１０では、その空き領域で新設される仮想命令の一つを、後述するネイティブ手続き命令（オペランドとしてエントリアドレスを受け取り、そのエントリアドレスにジャンプする処理を実行する）として予約する。

【００２３】また、このとき、実行制御部１１は、どの仮想命令について実行制御したのかを実行履歴取得部１２に通知する。

【００２４】一方、この通知を受けた実行履歴取得部１２は、この通知された仮想命令と一つ前に通知された仮想命令とをペアにして、その出現回数を実行履歴取得テーブル１７を用いてカウントする。図３は、実行履歴取得テーブル１７の一例を示す図であり、たとえば今回通知された仮想命令がＨでその一つ前に通知された仮想命令がＦであれば、実行履歴取得部１２は、Ｆ行Ｈ列（Ｆ，Ｈ）の値をインクリメントし、逆に、今回通知された仮想命令がＦでその一つ前に通知された仮想命令がＨであれば、実行履歴取得部１２は、Ｈ行Ｆ列（Ｈ，Ｆ）の値をインクリメントすることにより、各ペアの出現回数をカウントする。

【００２５】実行履歴取得部１２は、しきい値Ｓとサンプル期間Ｔとをもち、また、このしきい値Ｓを制御するための値ｓ１，ｓ２をもっている（ｓ１＜ｓ２）。そして、実行履歴取得部１２は、まず、しきい値Ｓをｓ１で初期化し、いずれかのペアの出現回数がしきい値Ｓを越えると、これらを１つの仮想命令として連結すべくその旨を仮想命令連結部１３に通知する。なお、実行履歴取得部１２は、サンプル期間Ｔが経過するごとに、しきい値Ｓにｓ１を加えていき、しきい値Ｓ＞ｓ２となったら実行履歴取得テーブル１７のすべてのカウント値を０で初期化するとともに、しきい値Ｓをｓ１で初期化し直す。これにより、仮想命令の連結をより適応的に行なうことを実現する。

【００２６】そして、仮想命令連結部１３は、実行履歴取得部１２から通知された仮想命令のペアを一つの仮想命令に連結すべく仮想命令列の再構成を実行する。以下、図４乃至図７を参照して、この仮想命令連結の一例を説明する。

【００２７】いま、仮想命令Ｇと仮想命令Ｒのペア（Ｇ，Ｒ）の出現回数がしきい値Ｓを越え、実行履歴取得部１２から仮想命令連結部１３に対してこれらの連結が通知されたものと想定する。図４は、ペア（Ｇ，Ｒ）の連結が行われる前の状態を示す図である。

【００２８】図４に示すように、連結前の状態では、アプリケーションプログラム２０の仮想命令列２１にはペア（Ｇ，Ｒ）が多数点在しており、一方、インタプリタ１０のディスパッチテーブル１６には、仮想命令Ｇおよび仮想命令Ｒともに仮想命令処理部１４の正規の処理部Ｇおよび処理部Ｒのエントリアドレスが格納されている。

【００２９】ペア（Ｇ，Ｒ）を連結する際、図５に示すように、仮想命令連結部１３は、まず、仮想命令Ｇの次が仮想命令Ｒかどうかをチェックする処理部Ｃ（Ｇ，Ｒ）と、仮想命令Ｇと仮想命令Ｒとを連結した処理部Ｊ（Ｇ，Ｒ）と命令書き換えのための処理部Ｗ（Ｇ，Ｒ）とを生成する処理部Ｇ（Ｇ，Ｒ）とを仮想命令処理部１４に生成し、ディスパッチテーブル１６に格納する仮想命令Ｇのエントリアドレスを処理部Ｇのエントリアドレスから処理部Ｃ（Ｇ，Ｒ）のエントリアドレスに書き換える。図５では、この処理を完了した時点での状態を示している。

【００３０】次に、この状態でプログラムカウンタ１２によりペア（Ｇ，Ｒ）の仮想命令Ｇが指し示されると、実行制御部１１によってディスパッチテーブル１６が参照され、書き換えられた処理部Ｃ（Ｇ，Ｒ）へのジャンプが行われることになる。

【００３１】処理部Ｃ（Ｇ，Ｒ）は、次の仮想命令が実行制御部１１からフェッチされ、その仮想命令がＲであれば処理部Ｇ（Ｇ，Ｒ）に、そうでなければ、処理部Ｇにジャンプする（処理部Ｇにジャンプする代わりに処理部Ｇのコピーを結合しておいてもよい）ものであり、ここでは、次が仮想命令Ｒであるため、処理部Ｇ（Ｇ，Ｒ）にジャンプする。

【００３２】一方、図６に示すように、処理部Ｇ（Ｇ，Ｒ）は、まず、処理部Ｇと処理部Ｒとを連結した処理部Ｊ（Ｇ，Ｒ）を仮想命令処理部１４に生成し、ディスパッチテーブル１６の空き仮想命令のプールから１つの仮想命令を取得して割り当てるとともに（これをＪ（Ｇ，Ｒ）とする）、このペアも実行履歴取得部１２のカウント対象とすべく実行履歴取得テーブル１７の行列に加える。

【００３３】次に、処理部Ｇ（Ｇ，Ｒ）は、インタプリトされているアプリケーションプログラム２０の仮想命令ペア（Ｇ，Ｒ）を書き換えるための処理部Ｗ（Ｇ，Ｒ）を仮想命令処理部１４に生成し、処理部Ｃ（Ｇ，Ｒ）から処理部Ｇ（Ｇ，Ｒ）への分岐を処理部Ｗ（Ｇ，Ｒ）への分岐に書き換える。

【００３４】最後に、処理部Ｇ（Ｇ，Ｒ）は、処理部Ｗ（Ｇ，Ｒ）へジャンプする（処理部Ｗ（Ｇ，Ｒ）は、処理部Ｇ，処理部Ｒを数値パラメータとして与えられれば実現できる定型的な処理なので、実行前に用意してもよい）。図６では、この処理を完了した時点での状態を示している。

【００３５】また、図７に示すように、ジャンプ先の処理部Ｗ（Ｇ，Ｒ）では、インタプリトされている仮想命令列２１のカレントのプログラムカウンタ１２が指し示している仮想命令ＧをＪ（Ｇ，Ｒ）に書き換え、また、次の仮想命令Ｒをオペランドに書き換え、処理部Ｊ（Ｇ，Ｒ）にジャンプする。そして、処理部Ｊ（Ｇ，Ｒ）にて、仮想命令ペア（Ｇ，Ｒ）が一括して実行される。図７では、この処理を完了した時点での状態を示している。

【００３６】以降、仮想命令列２１の仮想命令ペア（Ｇ，Ｒ）が現れる度に、処理部Ｃ（Ｇ，Ｒ）から処理部Ｗ（Ｇ，Ｒ）へのジャンプが行われ、仮想命令列２１

の書き換えと仮想命令ペア（Ｇ，Ｒ）の一括実行とが繰り返され、すべての仮想命令ペア（Ｇ，Ｒ）がいずれＪ（Ｇ，Ｒ）に書き換えられることになる。また、仮想命令Ｇの次に仮想命令Ｒ以外の仮想命令がくる場合は、処理部Ｃ（Ｇ，Ｒ）から処理部Ｇにジャンプして何ら問題なく処理される。さらに、その書き換え完了後であれば、仮想命令Ｇが重複するたとえば仮想命令ペア（Ｇ，Ｓ）についても、ディスパッチテーブル１６に格納する仮想命令Ｇのエントリアドレスを処理部Ｊ（Ｇ，Ｒ）のエントリアドレスから処理部Ｊ（Ｇ，Ｓ）のエントリアドレスに置き換えつつ、同様の処理を行うことが可能である。

【００３７】このように、このインタプリタのプログラム実行方法によれば、実行頻度の高い仮想命令列について、その実行に必要となるｎ回のループを１回のループで済むように動的に再構成を行うことにより、プログラムの起動時間を犠牲にすることなく、ＪＩＴコンパイラより単純な仕組みで同等の高速化を実現することを可能とする。

【００３８】また、この連結によって生成された仮想命令Ｊ（Ｇ，Ｒ）も、他の仮想命令とのペアの出現回数がカウントされ、アプリケーションプログラム２０の実行状況によっては、さらに他の仮想命令との連結が進められることになる。

【００３９】ところで、このように連結を進めていくと、実行頻度の高い部分の仮想命令列はいずれ１つの仮想命令となる。そこで、仮想命令連結部１３は、１つの仮想命令に連結しようとする元の仮想命令の数が、所定のしきい値を越えるか、あるいは基本ブロック全域となる場合に、その実行効率がより高速になるように、連結した処理単位に最適化をかける。ここでいう基本ブロックとは、プログラムを分岐命令と分岐命令の分岐先とで分割したときにできる必ずシーケンシャルに実行される各部分をいう。

【００４０】このとき、仮想命令連結部１３は、この仮想命令の処理部が基本ブロックの全域になっていれば、隣接する基本ブロックを調べ、隣接する基本ブロックも１つの仮想命令であって、かつ、これら基本ブロックの連続領域について、入口と出口とがそれぞれ１つの場合に、これら複数の基本ブロックの処理を一つの仮想命令として、この全体の処理について最適化した処理部を生成する。最適化された処理部は、もはやオペランド部の情報を必要としないので、連結された仮想命令のオペランド部の長さがターゲットのマイクロプロセッサの手続きのアドレス表現に必要な長さ以上になった場合に、この仮想命令を、前述したネイティブ手続き呼出命令（オペランドとしてエントリアドレスを受け取り、そのエントリアドレスにジャンプする処理を実行する）に書き換える。以下、図８および図９を参照して、この仮想命令書き換えの一例を説明する。

【００４１】いま、連結により新たに生成された仮想命令Ｊ１～Ｊ４が存在し、かつ、仮想命令Ｊ１と仮想命令Ｊ２、および仮想命令Ｊ３と仮想命令Ｊ４をさらに連結するものとする。そして、双方ともに、ネイティブ手続き呼出命令に書き換えるための条件を満たすものとする。図８は、この書き換え前の状態を示す図である。

【００４２】図８に示すように、書き換え前の状態では、アプリケーションプログラム２０の仮想命令列２１にはペア（Ｊ１，Ｊ２）とペア（Ｊ３，Ｊ４）とが点在しており、一方、インタプリタ１０のディスパッチテーブル１６には、仮想命令Ｊ１～Ｊ４それぞれに仮想命令処理部１４の新設の処理部Ｊ１～Ｊ４のエントリアドレスが格納されている。

【００４３】そして、図９に示すように、仮想命令連結部１３は、まず、仮想命令Ｊ１～Ｊ２を連結した処理部Ｊ（Ｊ１＋Ｊ２）と仮想命令Ｊ３～Ｊ４を連結した処理部Ｊ（Ｊ３＋Ｊ４）とを仮想命令処理部１４に生成し、次いで、仮想命令列２１に点在するペア（Ｊ１，Ｊ２）をオペランドとして処理部Ｊ（Ｊ１＋Ｊ２）のエントリアドレスをもったネイティブ手続き呼出命令に書き換え、また、仮想命令列２１に点在するペア（Ｊ３，Ｊ４）をオペランドとして処理部Ｊ（Ｊ３＋Ｊ４）のエントリアドレスをもったネイティブ手続き呼出命令に書き換える。図９では、この処理を完了した時点での状態を示している。

【００４４】これにより、連結された処理単位のさらなる高速化が図られ、また、Ｊ１～Ｊ４の実行頻度が低下することにより、結果として有限である空き仮想命令を有効利用することが可能となる。

【００４５】次に、図１０および図１１を参照して、このインタプリタ１０のプログラム実行方法の動作手順を説明する。

【００４６】このインタプリタ１０は、実行制御部１１が、プログラムカウンタ１５が指し示す仮想命令２１をロードし（図１０のステップＡ１）、同時に、実行履歴取得部１２が、仮想命令列の実行履歴を取得する（図１０のステップＡ２）。

【００４７】また、実行履歴取得部１２は、出現回数がしきい値を越える仮想命令列が存在しないかを監視し（図１０のステップＡ３）、しきい値を越える仮想命令列の存在を検出すれば（図１０のステップＡ３のＹＥＳ）、その仮想命令列の連結を仮想命令連結部１３に実行させる（図１０のステップＡ４）。

【００４８】一方、仮想命令連結部１３は、連結すべき仮想命令列のもとの仮想命令数がしきい値を越えるか、あるいは基本ブロックとなっているかを判断し（図１１のステップＢ１）、もとの仮想命令数がしきい値を越えるか、あるいは基本ブロックとなっていた場合（図１１のステップＢ１のＹＥＳ）、この連結すべき仮想命令列をネイティブ手続き呼出命令とすべく仮想命令列の再構

成を実行する（図１１のステップＢ２）。

【００４９】また、もとの仮想命令数がしきい値を越えず、かつ基本ブロックともなっていない場合（図１１のステップＢ１のＮＯ）、まず、仮想命令連結部１３は、空き仮想命令の取得を試み（図１１のステップＢ３）、もし取得できなければ（図１１のステップＢ４のＮＯ）、実行頻度の最も低い空き仮想命令を解放する（ステップＢ５）。この解放は、その空き仮想命令のために書き換えた仮想命令列２１およびディスパッチテーブル１６を元に戻すことによって実施する。その後、仮想命令連結部１３は、連結すべき仮想命令列をその取得した空き仮想命令とすべく仮想命令列の再構成を実行する（図１１のステップＢ６）。

【００５０】そして、実行制御部１１は、ディスパッチテーブル１６から取得した仮想命令処理部１４のエントリアドレスにジャンプし（図１０のステップＡ５）、その仮想命令のサイズ分だけプログラムカウンタ１２をインクリメントした後（図１０ステップＡ６）、上記の処理を繰り返す。

【００５１】このように、このインタプリタのプログラム実行方法によれば、実行頻度の高い仮想命令列について、その実行に必要となるｎ回のループを１回のループで済むように動的に再構成を行うことにより、プログラムの起動時間を犠牲にすることなく、ＪＩＴコンパイラより単純な仕組みで同等の高速化を実現することを可能とする。

【００５２】なお、前述の実施形態では、仮想命令列の連結を、まず空き仮想命令を用いて実行することを原則とし、状況によって、ネイティブ手続き呼び出し命令を用いる例を説明したが、リソース環境等によっては、当初よりネイティブ手続き呼び出し命令を用いて実行しても構わない。

【００５３】また、前述の実施形態では、ネイティブ手続き呼び出し命令を空き仮想命令を用いて設ける例を説明したが、アドレス呼び出し用の仮想命令が予め定義されている場合には、その既存の仮想命令を用いれば良い。

【００５４】ところで、仮想命令数が少なく、空き仮想命令が十分存在する場合は、小さなループのようなある程度まとまった部分の処理が１つの仮想命令となったり、ネイティブ手続き呼出命令となって、ＪＩＴコンパイラと同等の性能を得ることが可能であるが、空き仮想命令が十分存在しないときには、十分な効果が得られないおそれがある。そこで、実行履歴を採取して最適化する対象を、そのとき高速化したい手続きに限定する機能をさらに備えれば、たとえばプログラムの実行計画などに対応させて最適化の効果を得やすくすることが可能となる。

【００５５】さらに、メソッド、関数または処理単位等、最適化の対象ごとに異なるディスパッチテーブル１６を用いれば、このメソッド、関数または処理単位等ごとに固有の仮想命令連結を行うことが可能となる。

【００５６】

【発明の効果】以上詳述したように、この発明のインタプリタによれば、拡張性の確保等のために一般的に用意される空き仮想命令を利用して、プログラムの実行中に実行頻度の高い仮想命令列を動的に１つの仮想命令として再構成していくようにしたことから、実行頻度の高い仮想命令列について、その実行に必要となるｎ回のループを１回のループで済むように動的に再構成を行うことにより、プログラムの起動時間を犠牲にすることなく、ＪＩＴコンパイラより単純な仕組みで同等の高速化を実現することを可能とする。

【図面の簡単な説明】

【図１】この発明の実施形態に係るインタプリタのプログラム実行方法を説明するための概念図。

【図２】同実施形態のインタプリタが備えるディスパッチテーブルの構成を示す図。

【図３】同実施形態のインタプリタが備える実行履歴取得テーブルの構成を示す図。

【図４】同実施形態のインタプリタが実行する仮想命令連結の一例を説明するための第１の図。

【図５】同実施形態のインタプリタが実行する仮想命令連結の一例を説明するための第２の図。

【図６】同実施形態のインタプリタが実行する仮想命令連結の一例を説明するための第３の図。

【図７】同実施形態のインタプリタが実行する仮想命令連結の一例を説明するための第４の図。

【図８】同実施形態のインタプリタが実行するネイティブ手続き呼出命令への書き換えの一例を説明するための第１の図。

【図９】同実施形態のインタプリタが実行するネイティブ手続き呼出命令への書き換えの一例を説明するための第２の図。

【図１０】同実施形態のインタプリタのプログラム実行方法の動作手順を説明するための第１のフローチャート。

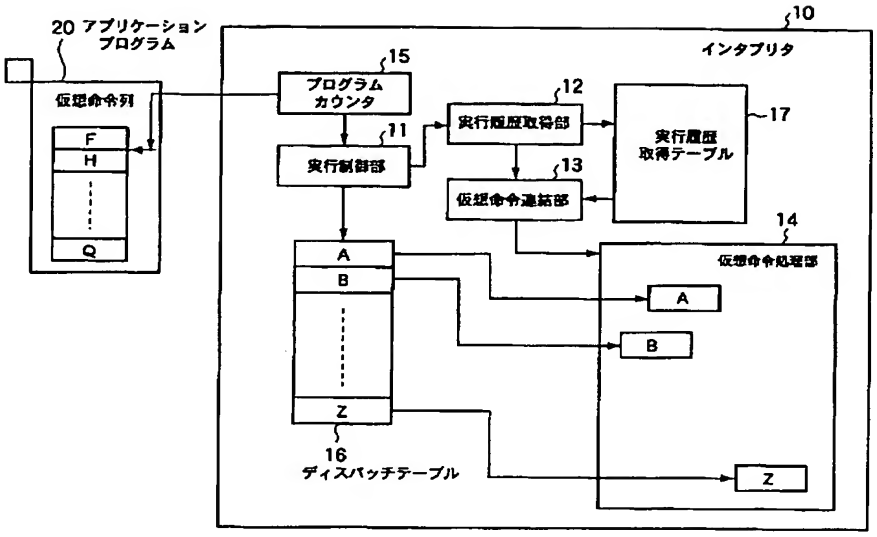
【図１１】同実施形態のインタプリタのプログラム実行方法の動作手順を説明するための第２のフローチャート。

【符号の説明】

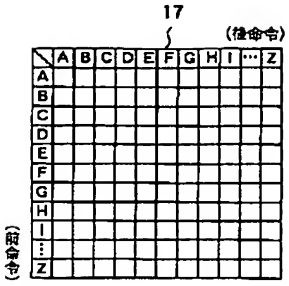
１０…インタプリタ
１１…実行制御部
１２…実行履歴取得部
１３…仮想命令連結部
１４…仮想命令処理部
１５…プログラムカウンタ
１６…ディスパッチテーブル
１７…実行履歴取得テーブル
２０…アプリケーションプログラム

２１…仮想命令列

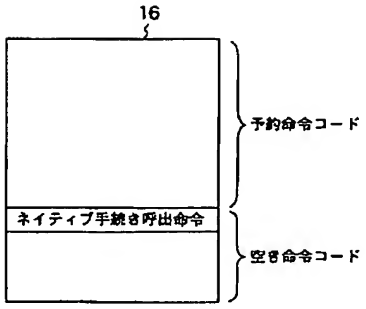
【図１】



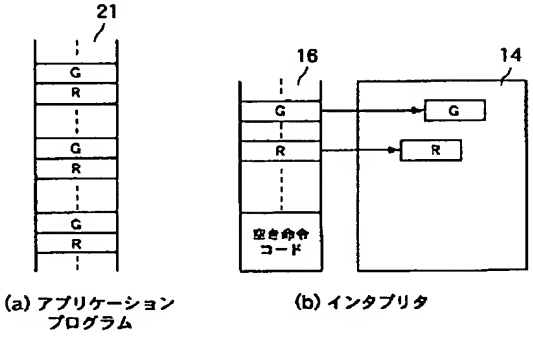
【図３】



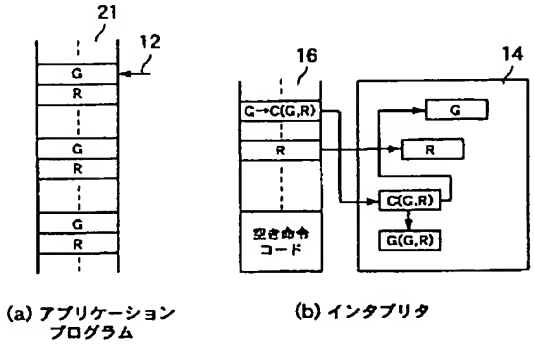
【図２】



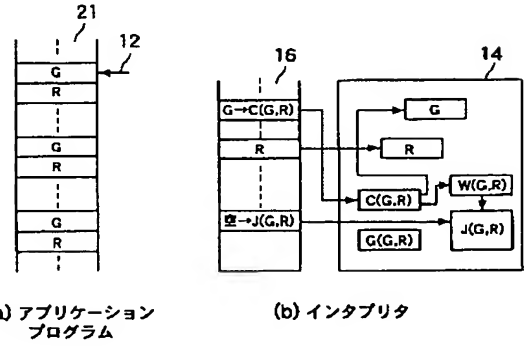
【図４】



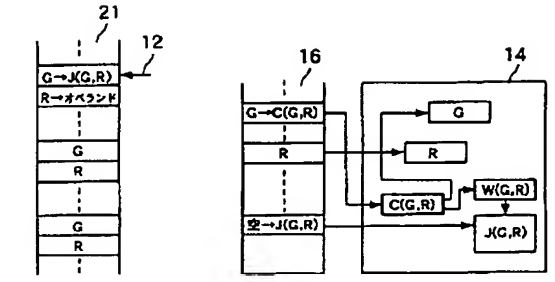
【図５】



【図６】

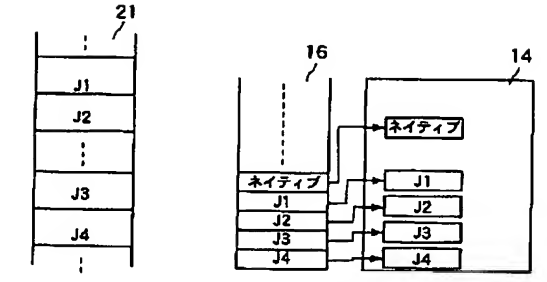


【図７】



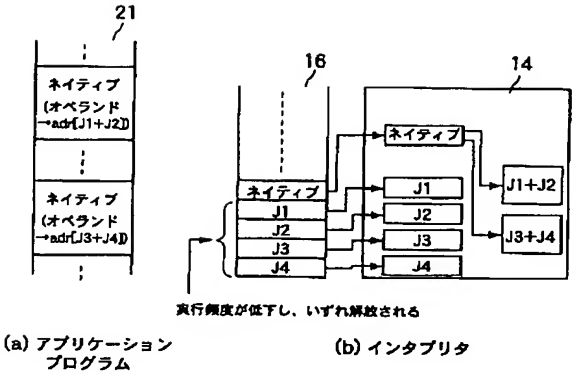
(a) アプリケーション プログラム　　(b) インタプリタ

【図８】



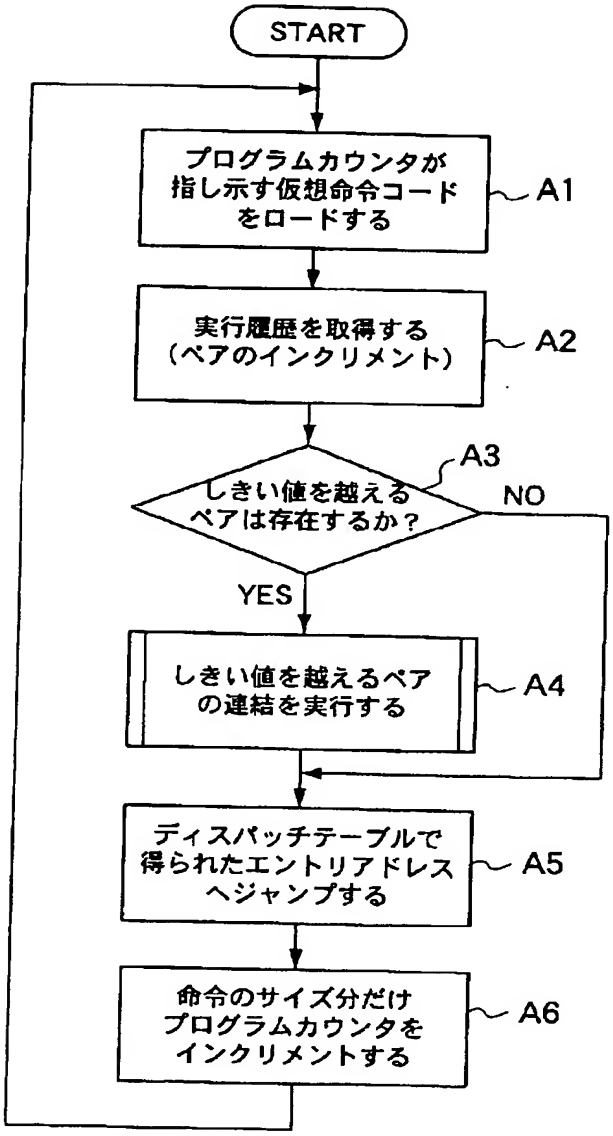
(a) アプリケーション プログラム　　(b) インタプリタ

【図９】

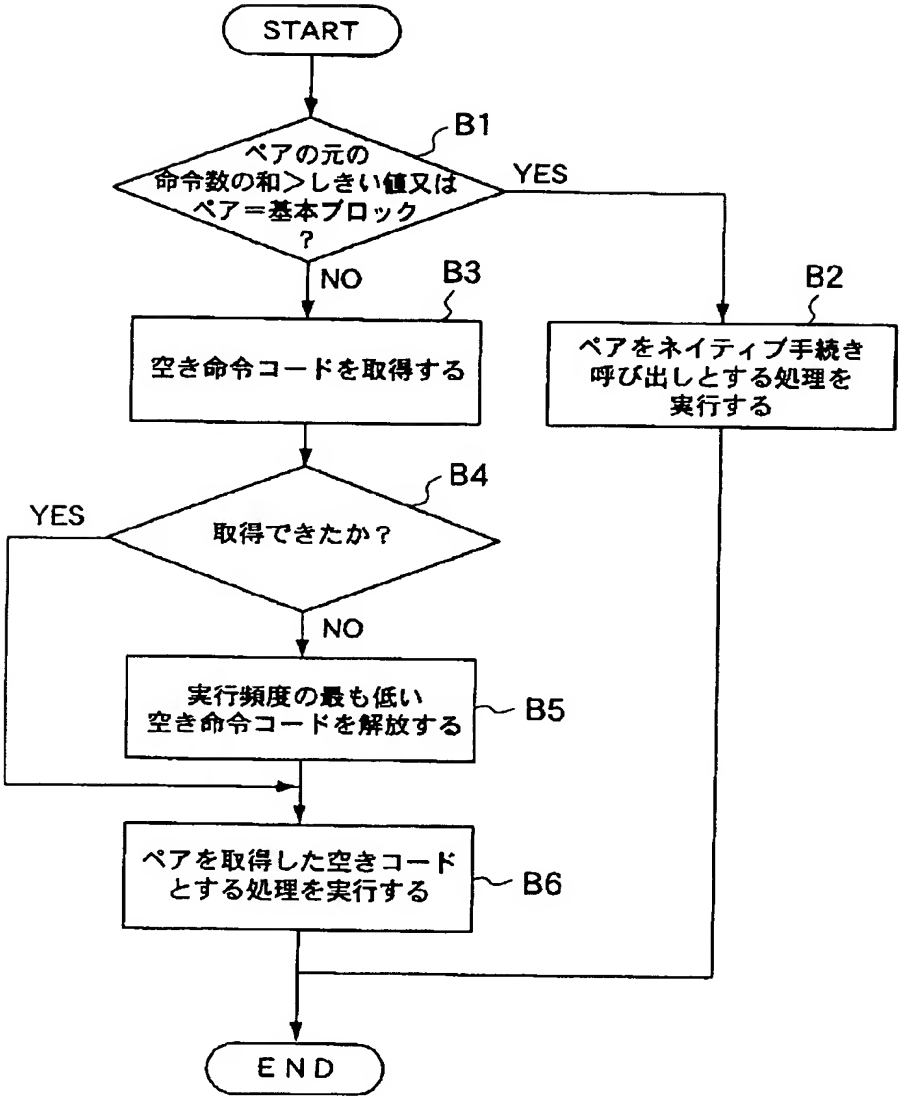


(a) アプリケーション プログラム　　(b) インタプリタ

【図１０】


START
プログラムカウンタが
指し示す仮想命令コード
をロードする
A1
実行履歴を取得する
（ペアのインクリメント）
A2
A3
しきい値を越える
ペアは存在するか？
NO
YES
しきい値を越えるペア
の連結を実行する
A4
ディスパッチテーブルで
得られたエントリアドレス
へジャンプする
A5
命令のサイズ分だけ
プログラムカウンタを
インクリメントする
A6

【図１１】


START
B1
ペアの元の命令数の和＞しきい値又はペア＝基本ブロック？
YES
NO
B3
空き命令コードを取得する
B2
ペアをネイティブ手続き呼び出しとする処理を実行する
B4
YES
取得できたか？
NO
実行頻度の最も低い空き命令コードを解放する
B5
ペアを取得した空きコードとする処理を実行する
B6
END